Trabalho Prático 2: Boosting

Aprendizado de Máquina 2024/1 UFMG

Nome: Isis Ferreira Carvalho Matrícula: 2020006663

1. Escolhas de implementação

Os seguintes passos foram seguidos durante a implementação do projeto:

- Tratamento dos dados

Fiz a leitura dos dados e verifiquei que, para trabalhar com o modelo Adaboost, seria necessário fazer um tratamento dos dados de entrada. Nesse sentido, apliquei *OneHotEncoding* nas colunas que alimentariam o modelo, transformando as 9 variáveis categóricas, que representavam o conteúdo das posições do tabuleiro de Tic-Tac-Toe (x, o ou b), em 27 variáveis binárias que representam se, em uma dada posição do tabuleiro, há um dado conteúdo. Com esse processo, os dados estão corretamente preparados para serem passados para o modelo como entrada.

- Implementação da classe **BoostingAlgorithm**

A classe **BoostingAlgorithm** contém toda a lógica de inicialização, treino e teste do modelo. Um objeto desta classe, ao ser inicializado, precisa informar os dados de entrada e as *labels* destes dados, assim como os índices dos dados a serem usados no treino e no teste – de forma a ser compatível com validação cruzada – e o número de iterações que o algoritmo executará (que são o número de stumps a serem utilizados. Alguns atributos do objeto são inicializados também durante esse processo de inicialização, como os pesos iniciais (W) dos dados de treino e as listas que guardarão os *stumps* de cada iteração, e os seus respectivos alfas.

A função *train_model* é responsável pelo treino do modelo. Para cada uma das iterações do algoritmo, um stump, que consiste em um *DecisionTreeClassifier* (com *max_depth* = 1) da biblioteca *scikit-learn*, é inicializado e ajustado aos dados ponderados pelo peso da iteração presente. Após esse *fit*, o erro de classificação e o alfa são calculados e os pesos são atualizados. Esse processo continua até que todas as iterações sejam feitas.

A função *test_model* executa o modelo resultante do treino nos dados de teste, somando as predições de cada stump ponderadas pelo alfa correspondente, e aplicando a função sinal após essa soma, obtendo a predição final do modelo. Com essa predição, é possível avaliar o quanto o modelo acertou e errou. A especificação do trabalho recomenda a taxa de erro simples para tal tarefa, mas para ter uma noção melhor da performance dos modelos avaliados, eu preferi utilizar a acurácia (que é justamente 1 menos essa taxa de erro) para avaliar a performance dos modelos. A acurácia é retornada ao final desta função.

- Experimentos

Como especificado, a avaliação do modelo deveria ser feita utilizando validação cruzada com 5 partições. Isso foi feito utilizando o recurso KFold, da biblioteca *scikit-learn*. Variamos o número de iterações entre 10 e 500, andando de 10 em 10 unidades. Assim, passamos

por 50 valores de número de iterações diferentes. Uma observação importante é que, para esse número de testes, o código demorou menos de 4 minutos para completar os experimentos. Não foram testados valores maiores que 500, pois foi observado empiricamente que, para esses valores, o modelo ficava flutuando perto da acurácia máxima.

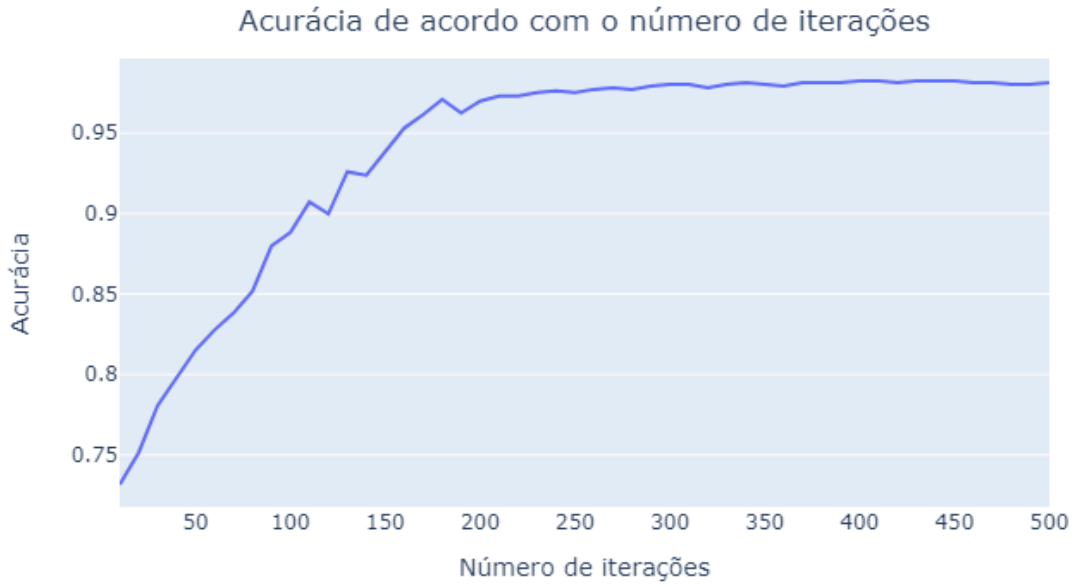
O *loop* de execução dos experimentos consiste na seguinte lógica:

- ➔ Para cada número de iterações **n**:
  - ◆ Para cada iteração da validação cruzada:
    - Objeto da classe BoostingAlgorithm é inicializado com **n** e os índices do fold correspondente à iteração
    - Modelo é treinado nos dados de treino
    - Modelo é testado e salvamos sua acurácia em uma lista
  - ◆ Obtemos acurácia média do modelo, fazendo média das acurácias nos folds
  - ◆ Salvamos este resultado atrelado ao número de iterações em uma lista

A lista mencionada ao fim dá origem a um DataFrame, que é salvo como um arquivo de nome *results.csv* ao fim dos experimentos.

2. Resultados dos experimentos

Para avaliar os modelos obtidos neste trabalho, projetamos os resultados dos experimentos em um gráfico de linha utilizando a biblioteca *plotly*. Como dito anteriormente, após 500 iterações, o modelo para de apresentar melhora significativa (e começa a flutuar perto de um valor de convergência). O gráfico que mostra a acurácia alcançada nos experimentos se encontra a seguir:



No gráfico acima, vemos que a acurácia vai aumentando aos poucos à medida que o número de iterações aumenta. O aumento é mais acentuado no começo, e vai se

estabilizando por volta de 250 iterações, ponto no qual atingimos um nível de convergência de acurácia. A maior acurácia alcançada é de 98.22%, com 400 iterações. Acima de 400 iterações, a acurácia não aumenta mais, mas também não chega a cair muito, mostrando bem que o modelo não sofre de *overfit*, visto que é incapaz disso por construção. Assim, como o número de iterações é diretamente proporcional ao custo computacional do modelo, já poderíamos ter parado os experimentos em 400 iterações, ou, de uma forma generalizável para outros problemas, podemos parar assim que a acurácia não aumenta mais.